EV316937305

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

of

Jason P. Chalecki
Kelvin S. Yiu
and
Prakash Sikchi

for

Rendering An HTML Electronic Form By Applying XSLT To XML Using A Solution

ATTORNEY'S DOCKET NO. MS1-1699US

Rendering An HTML Electronic Form By Applying XSLT To XML Using A Solution

RELATED APPLICATIONS

[0001] This application is a continuation-in-part of US Patent Application Serial No. 10/610,504, filed on June 30, 2003, titled "Declarative Solution Definition", which is incorporated in its entirety by reference.

TECHNICAL FIELD

[0002] This invention generally relates to writing data to, and viewing data in, an electronic form that is related to a hierarchical data file, and more particularly, to a solution for a hierarchical data file that declaratively defines features of the hierarchical data file, where an electronic form can be rendered by using the solution to transform the hierarchical data file such that the electronic form can be used to write data to, and view data in, the hierarchical data file.

BACKGROUND

[0003] Extensible markup language (XML) is increasingly becoming the preferred format for transferring data. XML is a tag-based hierarchical language that is extremely rich in terms of the data that it can be used to represent. For example, XML can be used to represent data spanning the spectrum from semi-structured data (such as one would find in a word processing document) to generally structured data (such as that which is contained in a table). XML is well-suited for many types of communication including business-to-business and client-to-server communication. For more information on XML, XSLT, and XSD (schemas), the reader is referred to the following documents which are the work of, and available from the W3C (World Wide Web consortium): XML

Schema Part 2: Datatypes; XML Schema Part 1: Structures, and XSL Transformations (XSLT) Version 1.0; and XML 1.0 second edition specification.

[0004] One of the reasons that data files written in XML are often preferred for transferring data is that XML data files contain data, rather than a combination of data and the software application needed to edit the data. Using XML is advantageous, as opposed to a binary data format, because it is all in a text format which makes it possible to view and edit a raw XML file using standard applications like an Internet browser and a text editor.

[0005] In some applications, in order to edit an XML data file, a user typically must interactively install a solution software application used to access, view, and edit the data file. When the user is online, the user's computer can run a host application capable of accessing the Internet, such as Microsoft® Internet Explorer®, which can silently discover and deploy a solution, which can be written in XSLT, where the solution enables the user to author and access an XML data file. Alternatively, some XML files can be directly viewed in an Internet browser so that a specific application may not be needed for this specific purpose.

[0006] An application and/or information to be put into XML data files can be collected electronically using, for example, Internet (e.g., web) or online electronic forms. Tools for using the electronic forms must be custom built applications or must use proprietary electronic forms tools. Using these custom applications and proprietary electronic forms tools causes at least four significant problems. The first problem is that the process of gathering information electronically can be inefficient. Inefficiencies occur for several reasons. One reason is that data entry personal can provide inconsistent data. Data inconsistencies occur where information is missing or provided in different formats. As

such, those responsible for gathering, analyzing or summarizing the data must go back and reconcile the information provided. Another data inconsistency occurs because people provide information to one person who then inputs the data into some sort of electronic form or tool. Because the data has to be re-typed by someone who isn't the subject matter expert, this process leads to inaccurate data and to an inefficient data gathering process.

[0007] A second problem with using custom applications and proprietary electronic forms tools is that the resultant electronic forms or documents aren't very easy to use and can be extremely inflexible. Many conventional electronic forms do not provide a rich, traditional document editing experience. Without tools or features such as rich text formatting and spell checking, these forms can be hard to use. And as a result, people don't use them as frequently as they should – leading to loss of invaluable organizational information. Additionally, conventional electronic forms are static in nature, where users have to fit their information into a set number of fields and don't have the ability to explain the context behind the information. What this means is that the information provided is often incomplete. As such, the person who consumes the information has to go back to the users to find out the real story behind the data.

[0008] A third problem with using custom applications and proprietary electronic forms tools is that they require considerable expense in order to build a solution that enables a user to author and access data using an electronic form or document corresponding to the solution. Such conventional electronic forms or documents can be difficult to modify because they were built for a specific purpose and required development work. Validation of data entered in conventional electronic forms for documents requires a developer to write code or script. Additionally, data entry personnel must be trained on

how to use the conventional electronic forms for documents. Moreover, once the data is collected for an organization using the conventional electronic forms, the data is difficult to re-use or re-purpose elsewhere in the organization because the collected data is locked into proprietary documents and data formats. To reuse this data, the organization must invest in significant development work to extract the data from the appropriate sources and translate the data from one format to another.

[0009] Given the foregoing, it would be an advantage in the art to provide a solution that can be discovered for a data file by which electronic forms can be used to enter data into and view data in the data file, where the solution addresses the foregoing problems.

SUMMARY

The following description and figures describes a solution for an XML document. In one implementation, instructions are received to open an eXtensible Markup Language (XML) document. The XML document is searched to locate a processing instruction (PI) containing a href attribute that points to a URL. The solution is discovered using the URL in the PI. The XML document is opened with the solution. The solution includes an extensible stylesheet language (XSLT) presentation application and a XML schema. The XML document can be inferred from the XML schema and portions of the XML document are logically coupled with fragments of the XML schema. The XSLT presentation application is executed to render a Hypertext Markup Language (HTML) electronic form containing data-entry fields associated with the coupled portions. Data entered through the data-entry fields can be validated using the solution. Other

implementations are disclosed for discovering a solution and rendering an HTML electronic form by applying XSLT to XML using the solution.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The detailed description is described with reference to the accompanying figures in which the same numbers are used throughout the disclosure and figures to reference like components and features. Series 100 numbers refer to features originally found in Fig. 1, series 200 numbers refer to features originally found in Fig. 2, and series 300 numbers refer to features originally found in Fig. 3, and so on.

[0011] Fig. 1 illustrates a communications network and a system capable of implementing a method for silently discovering and deploying a solution that corresponds to a data file having a hierarchical arrangement of a plurality of nodes, where the method includes use of the solution with an interactive tool that enables a user to enter data into the data file, to edit data in the data file, and to view data in the data file.

[0012] Fig. 2 illustrates an exemplary screen having a display area depicting a rendered blank electronic form and an incomplete view of a rendered form of a hierarchical data file, where both the blank electronic form and the rendered form of the hierarchical data file correspond to the hierarchical data file, and where the electronic form can be used by a user to enter data into the hierarchical data file, to edit data in the hierarchical data file, and to view data in the hierarchical data file.

[0013] Fig. 3 illustrates a block diagram have components for an XML solution corresponding to an XML document.

[0014] Fig. 4 is a block diagram that illustrates an exemplary collection of files that make up an electronic form template, where an application to use the electronic form template is invoked when a user navigates to an XML document.

[0015] Fig. 5 is a flow diagram illustrating exemplary relationships between design components for an electronic form application, runtime components for using an electronic form designed using the design components, and solutions components that are preexisting electronic forms that can be used with the electronic form application.

[0016] Fig. 6a is a flow diagram illustrating an exemplary process to deploy a form template in which a user opens a form of a certain type to automatically download the corresponding latest version of a form template that is stored on the user's machine so that the user can use the form template when not connected to a network.

[0017] Fig. 6b is a flow diagram illustrating an exemplary process to deploy a form template by installation directly on a user's computing device, where the form template can be packaged as an executable module.

[0018] Fig. 7 is a flow diagram of an exemplary process for editing of data files while online or offline.

[0019] Fig. 8 illustrates the exemplary screen display depicting the electronic form seen in Fig. 2, where data has been entered into a plurality of data-entry fields of the electronic form.

[0020] Fig. 9 is a flow diagram of an exemplary process for real-time validation of data for a data file.

[0021] Fig. 10 illustrates an exemplary screen display showing an electronic form with a data-entry field having an invalid entry and a dialog box.

[0022] Fig. 11 illustrates an example of a computing environment within which the solutions, software applications, methods and systems described herein can be either fully or partially implemented.

DETAILED DESCRIPTION

[0023] The following disclosure describes a way to access data files, either when online or when offline, using an electronic forms application. The disclosure also addresses the definition of interactivity and authoring experiences, connections to external data sources, and other functionalities with respect to the electronic forms application. If a user has opened a data file first online, or if the system has otherwise received the data file's solution, the electronic forms application can silently discover and deploy the data file's solution so that the user can edit data in the data file when offline. As used herein a solution and a form template for an electronic form are equivalent. The data file's solution declaratively defines aspects a data file such as its elements, attributes, and values, as will be discussed below. The electronic forms application allows a user to simply select a data file to open and the electronic forms application will open the data file with a discovered and deployed solution. The user need not discover, select, or even be aware that the data file requires a solution for the data file to be edited. After selecting the data file to open, the user can then edit and access the data file in a way very similar to how it would act and appear had the user opened the data file while online.

[0024] Data Files, Solutions, and Host Applications

Data files, their solutions, and a host application work together to allow a user to open and edit the data file. Data files contain little or no operable code, whereas a solution file contains presentation and logic applications. The presentation and logic applications of a solution file or solution declaratively define aspects a data file such as its elements, attributes, and values. The elements, attributes, and values that are declaratively defined can include a schema for the data file, one or more views that can be used for viewing and entering data in the data file, a manifest of one of more files that enable contextual editing of the data file, and one of more user interfaces that can be used

with the one or more views and can include functional components such as toolbars, menu bars, buttons, or task panes. Other declaratively defined elements, attributes, and values include a usage of specific event handlers and/or specific error handlers, and a definition of one or more back-end servers to which connectivity is available.

[0025] The elements, attributes, and values can also be programmatically defined in addition to the foregoing declarative definition. Specifically, the programmatic definition can be written in a programming code using a scripting language and can include validation rules for data entry with respect to the data file, custom error processing, implementations of data routing (data submission), and algorithms for connecting programmatically to databases, Web services or other back-end systems.

[0026] Editing a data file requires a solution. If a user tries to open a data file without a solution, the user could get an error or perhaps a flat list of the data in the data file. In order to view and edit the data file, the data file's solution is needed. As such, a solution has a corresponding solution application for the data file. The solution application is one or more files that, when installed, are used to enable a user to view, access, and edit the data file.

[0027] In addition to the data file and its solution, a host application is needed. This application works to enable the solution to function fully. In this description, an electronic forms application is described, which is capable not only of acting as a host application (allowing a solution to function properly), but can also allow a user to open a data file without actively finding and installing the data file's solution.

[0028] For discussion purposes, the system and method described herein are described in the context of a single computer, a communications network, a user-input device, and a

display screen. These devices will be described first, followed by a discussion of the techniques in which these and other devices can be used.

[0029] Exemplary Architecture

Fig. 1 shows an exemplary architecture 100 to facilitate online and offline editing of data files. This architecture 100 includes a computing system 102 connected to a communications network 104. The system 102 is configured to go online and communicate via the communications network 104 to gain access to non-local information sources, such as sources on an intranet or global network. Alternatively, the system 102 can remain offline, where it utilizes local resources without communicating over the communications network 104.

[0030] The computing system 102 includes a user-input device 106, a display 108 having a screen 110, and a computer 112. The user-input device 106 can include any device allowing a computer to receive a user's preferences, such as a keyboard, a mouse, a touch screen, a voice-activated input device, a track ball, and the like. With the user-input device 106, a user can edit a data file by adding or deleting information within a data-entry field on an electronic form, for instance. The user can use the display 108 and the electronic form on screen 110 to view the data files. An electronic form, with respect implementations of an electronic forms application 122 described herein, is a document with a set of controls into which users can enter information. Electronic forms can contain controls that have features such as rich text boxes, date pickers, optional and repeating sections, data validation, and conditional formatting. An example of the electronic forms application 122 is the InfoPath™ software application provided by Microsoft Corporation of Redmond, WA, USA, through the Microsoft Office™ software application.

[0031] The computer 112 includes a processing unit 114 to execute applications, a memory 116 containing applications and files, and a network interface 118 to facilitate communication with the communications network 104. The memory 116 includes volatile and non-volatile memory, and applications, such as an operating system 120 and the electronic forms application 122. The memory 116 also includes a solution 124 for a data file 126. The solution 124 is located locally in the memory 116, but often has a different original source, such as a source on the communications network 104. The solution 124 contains one or more files and folders, such as a presentation folder 128, a logic file 130, and a list file 132. The presentation folder 128 includes a rendering file 128a and a transformation file 128b. The components of solution 124 will be discussed in greater detail below.

[0032] The electronic forms application 122 facilitates offline editing of the data files 126 and is executed by the processing unit 114. The electronic forms application 122 is capable of acting as a host application and enabling a user to open the data file 126 without actively finding and installing the data file's solution 124. Without any user interaction, other than the user attempting to open the data file 126, the electronic forms application 122 discovers and installs the data file's solution 124. Thus, the user does not have to do anything but request to open the data file 126. The user does not have to discover the data file's solution 124. The user does not have to install the data file's solution 124. This silent discovery and deployment allows the user to view, edit, and otherwise interact with the data file 126 with just a single request. In addition, the electronic forms application 122 can provide security offline similar to the security that the user typically enjoys when running a solution online.

[0033] A view of the data file 126 can be depicted on screen 110 through execution of the data file's solution 124. The solution 124 contains one or more applications and/or files that the electronic forms application 122 uses to enable a user to edit the data file 126. To edit the data file 126 in a user-friendly way, the data file's solution 124 contains the presentation folder 128, which includes an electronic form. This presentation folder 128 is a container for components that, when used, give the user a graphical, visual representation of data-entry fields showing previously entered data or blank data-entry fields into which the user can enter data. Data files often have one solution but each solution often governs multiple data files.

[0034] Fig. 2 shows an electronic form 200 entitled "Travel Itinerary", which is generated by the solution 124. This travel itinerary form 200 contains data-entry fields in which a user can enter data. These data-entry fields map to the data file 126, so that the data entered into the form are retained in the data file 126. Fig. 2 shows a graphical representation of the data file 126 as a data file tree 202. The data file tree 202 shows icons representing nodes of the data file 126. Many of these nodes correlate to data-entry fields shown in the travel itinerary rendered form 200. For instance, a trip start date node 204 correlates to the trip start date data-entry field 206. Thus, data entered by a user into the trip start date data-entry field 206 can be stored in the trip start date node 204 of the data file 126.

[0035] The solution 124 presents the travel Itinerary form 200 but also contains the logic file 130 that governs various aspects of the travel Itinerary form 200 and the data file 126. In a data-entry field 206, for instance, the solution 124 can be configured so as to present the data-entry field 206 in a variety of visual appearances, such as by providing a white box within a gray box, by providing a description of the data desired with the text "Start

Date", and by providing logic that requires the user to pick a date by clicking on a calendar icon. Thus, if the user attempted to enter letters, the logic file 130 of the solution 124 would not permit the user's entry. The solution 124 could reject it and inform the user of the problem, such as with a sound, flashing error signal, pop-window, or the like.

[0036] The logic file 130 is employed in the solution 124 to ensure that the right kind of data is being entered and retained by the data file 126. A user's business manager attempting to reference dates with a reference number, for instance, would like the solution 124 to have dates in the data-entry field 206, otherwise, the manager may not be able to determine how a travel itinerary should be handled if the date is entered incorrectly because it contains letters.

[0037] Similarly, suppose a business manager wants the travel date for a trip. To require this, the logic file 130 of travel itinerary form 200's solution 124 could be constructed to require a date to be entered into the date data-entry field 206. The logic file 130 can be internal to the solution 124. The logic file 130 can also be a schema, such as an XML schema. Here, the XML schema is a formal specification, written in XML, that defines the structure of an XML document, including element names and rich data types, which elements can appear in combination, and which attributes are available for each element.

[0038] A solution can govern multiple data files. The exemplary travel itinerary form 200, for example, allows one or more users to fill out many different trips. Each time a user fills out a travel itinerary form 200, the system 102 can create a separate data file for that trip. Often, a user will create many different data files having the same solution. For each data file edited after the first, the system 102 is likely to have the appropriate solution stored in the memory 116. Thus, if a user previously opened a first data file and

later attempts to open a second data file, both of which utilize the travel itinerary form 200 solution, the electronic forms application 122 can silently discover and deploy the travel itinerary form 200 solution to enable the user to edit the second data file. How the electronic forms application 122 discovers and deploys solutions will be discussed in greater detail below.

[0039] A solution can be one file or contain many files, so long as the files used to edit data files it governs are included. In one implementation, the solution 124 of Fig. 1 includes the listing file 132, which is a manifest of all of the other files in the solution 124 and contains information helping the electronic forms application 122 to locate them. In an alternative implementation, this information can be inside a 'manifest file' so as to accomplish a similar function. The logic file 130 and presentation folder 128 can be joined or separate. The presentation folder 128 helps the electronic forms application 122 present or give a view of a form enabling entry of data into the data file 126, such as a visual representation of the data file 126 by the travel itinerary form 200 electronic form. In some implementations, the presentation file is an XSLT file, which, when applied to an XML data file, generates a XHTML (eXtensible Hyper-Text Markup Language) or HTML (Hyper-Text Markup Language) file. XHTML and HTML files can be used to show a view on the screen 110, such as the travel itinerary form 200 of Fig. 2.

[0040] A solution, such as the solution 124, can also include various files or compilations of files, including a manifest file setting forth names and locations for files that are part of the solution 124. The files within the solution 124 can be packaged together, or can be separate. When separate, the list file 132 acts as a manifest of the files within the solution 124. In one implementation, the list file 132 can also include other information, such as definitions, design time information, data source references, and the like. In

another implementation, file list information can be part of a manifest that stores the file list information. When the files are packaged together, the electronic forms application 122 can simply install and execute the packaged solution file for a particular data file. When not packaged, the electronic forms application 122 can read the list file 132, find the listed files, and install and execute each of the listed files for the particular data file.

[0041] Like solutions, data files can come in various types and styles. As mentioned above, data files can be written in XML or some other mark-up language, or can be written in other languages. Most data files, however, do not contain extensive logic and other files or code. One of the benefits of having data files separate from their solutions, is that it makes the data within them easier to mine. Because the data files are separate from their solution, the electronic forms application 122 makes them easy to open and edit by silently discovering and deploying the solution for the data file.

[0042] Data files also are typically concise and data-centered so that the data they contain can be more easily accessed or manipulated by multiple software applications, including software not typically used in a solution, such as an application that searches for a particular type of data and compiles that data into a report. A non-typical application, for example, could be one that compiles a report of all of the travel itineraries required to be mailed by a certain date by searching through and compiling the data entered into data files through the required data-entry field 206 of the travel itinerary form 200 electronic form.

[0043] The above devices and applications are merely representative, and other known devices and applications may be substituted for or added to those shown in Fig. 1. One example of another known device that can be substituted for those shown in Fig. 1 is the device shown in Fig. 11.

[0044] Fig. 3 depicts a variety of components that can be used by the electronic forms application 122 described herein. The electronic forms application 122 allows a user to design, view and complete an electronic form. The electronic forms application 122 also allows the user to enter and change data in a data file corresponding to the electronic form, where the electronic form corresponds to the data file for which there is a corresponding solution.

[0045] A plurality of XML solution files 300, seen in Fig. 3, represent a collection of files that are used to implement an electronic form processed by implementation of the electronic forms application 122 described herein. File types can include HTML, XML, XSD, XSLT, script, and other file types that are necessary to support the functionality of the electronic form. As seen in Fig. 3, the XML solution files 300 include an XML solution 302 and an XML document 304. In general, however, XML documents can be kept separate from the XML solution 302. A solution definition file 306 is the "hub" of the electronic forms application 122. The solution definition file 306 contains a declarative definition of some solution features and a manifest of all the solution files including XML document templates 308, business logic files 310, view definition files 312, and view interactivity files 314. In a different implementation the view interactivity files 314 can be included with the solution definition 306. The XML document templates 308 are prototypes for each document type used in the electronic forms application 122. The XML document templates 308 are used to create new XML documents. Alternatively, an XML document template 308 can be a file that contains sample data that is displayed in the fields of an electronic form before a user fills out the electronic form. One of the XML Documents can be identified in the solution definition file 306 as a default document for the electronic forms application 122. The business logic files 310

contain validation information associated with a XML document type, which is discussed below with respect to Figs. 9-10.

[0046] Having discussed general data files and solutions for viewing, editing, authoring general data files, above, a discussion will now be made specifically about XML documents and solutions. The view definition files 312 are XSLT files associated with each XML document 304. The view definition files 312 define multiple layouts and view logic for the XML document 304. The view interactivity files define a contextual user interface (UI) as well as behavior of some XSLT views. Each XSLT view is an electronic form-specific display setting that can be saved with an XML solution 302 and applied to form data when the electronic form is being filled out. Users can switch between views to choose the amount of data shown in the electronic form.

[0047] The XML solution files 308-314 are a collection of files that are used to implement an electronic form as further described herein. File types can include HTML, XML, XSD, XSLT, script, and other file types that are helpful in supporting the functionality of the electronic form.

[0048] The information (e.g., data) that is collected in a single electronic form, further described herein, can be used by many different systems and processes. The usefulness of this information is due the storage of the information as XML. The electronic forms application 122 can communicate with ADO (ActiveX Data Objects) databases and XML Web services in a bi-directional manner, enabling the information stored in databases and servers to populate data-entry fields within electronic forms.

[0049] The solution definition file 306 is described above as the hub file because it can contain all the information to determine how the XML document 304 is presented to the end-user for editing of the information and ways to interact with the information, such as

by navigating to different views of the information, modifying content or creating new content, etc. The solution definition file 306 can reference secondary files, some of which have types of data corresponding to XML standards, like XSD files that are used for schema information, like XSLT files that are used to specify the direct visual presentation of the XML data as views, and like XSLT style-sheets that are used to transform the XML document that has been loaded into HTML for visual presentation.

[0050] The solution definition file 306 uses XML vocabulary in an interoperable file format for XML solution 302. A declarative specification for contextual interactivity with XML data, via an XSL generated view, is included in the solution definition file 306. Additionally, the solution definition file 306 contains or references information about all other files and components used within a form, including files that contain instructions for user interface customizations, XML schemas, views, business logic, events, and deployment settings. Here, an event is an action recognized by an object, such as a mouse click or key press, for which a response by the electronic forms application 122 can be defined. An event can be caused by a user action or it can be triggered by the system. An event can be caused by a statement of business logic that is written in any supported programming language (e.g., a Visual Basic statement).

[0051] The solution definition file 306 is an XML document that can also express information that does not correspond to any XML standard via an XML syntax that allows declarative specification of XML solution 302. Various features can be provided to the end-user by the solution definition file 306.

[0052] Once feature that is provided by the solution definition file 306 is the ability to define views and commands/actions that can be made available via secondary user interface as well as how the views' availability will be determined contextually, including

the interactivity that is to be made available in the views. Here, the secondary user interface (UI) referred to is a context driven UI that offers menus and toolbars. Another feature provided by the solution definition file 306 is the types of binding between the views and the underlying XML data so that data entered can be saved in an XML file. The binding here refers to a logical connection from a control (e.g., a data entry field on an electronic form) to a field or group in a data source of an data file so that data entered into the control is saved. Here, a data source is a collection of fields and groups that define and store the data for an electronic form being processed with the electronic forms application 122. Controls in the form are bound to the fields and groups in the data source. Specifically, a control is a graphical user interface object, such as a text box, check box, scroll bar, or command button, that lets users control the electronic forms application 122. Controls can be used to display data or choices, perform an action, or make the user interface easier to read. Conversely, when a control is unbound, it is not connected to a field or group, and so data entered into the control will not be saved.

[0053] A still further feature is the availability of structural or text-level editing operations on the XML data. Yet another feature includes validations (e.g., XML data validation), event handlers associated with the electronic form as a whole, and business logic associated to individual nodes of the XML Document Object Model (DOM) representing the form, where the business logic attaches to the electronic form as a whole, including the association of 'business logic' script with user actions. A still further feature includes simple workflow and information about routing of the XML Document 304. The availability to use global metadata information about the XML Document 304, including deployment/publishing information, is another feature. Another feature provided by the solution definition file 306 is the availability of default XML data that

can be used when creating a new XML Document 304. A unique identifier for the electronic form is an other feature. A still further feature is the availability of optional schema for the XML DOM that comprises the electronic form.

[0054] The electronic forms application 122 described herein enables users to process XML documents 304. XML documents 304 are partitioned into classes or document types, based on their schemas. XML documents 304 need business logic 310 and data flow as required by the XML data solution 302 they are part of. This business logic 310 and data flow information lives outside of the XML document 304 in XML solutions 302. As such, the XML solution is a factory for given classes of XML Documents. The XML solution 302 defines the layouts and the editing behavior of the XML Documents 304. The XML solution 302 enforces data consistency and provides routing information. The electronic forms application 122 described herein may work with one or more types of XML documents 304 and provide a portal and a list views for collection of XML documents 304. XML solutions 302 can be stored on the client, such as in computer 112, and can be made available offline.

[0055] Fig. 4 depicts an example of several files that make up or are referred to by an electronic form template 420. The electronic form template 420 can be a manifest of all files used by the electronic forms application 122 described herein. The electronic form template 420 can be invoked when a user navigates to an XML document or when a new XML document is to be created. The electronic form template 420 is a collection of files that declaratively defines the layout and functionality for an electronic form. Electronic form templates 420 can be stored either as a single compressed file or as a folder of files. This collection of files is required for any XML document to be opened and filled out in the electronic forms application 122. The electronic form template 420 includes an XML

schema definition 406, a form definition 404, and one or more XSLT filed 408 for defining views.

[0056] The XML schema file 406 defines the structure for the XML created when filling out the electronic form. The one or more XSLT files 408 define different views. The definition of different views can be written in a language, such as XSLT, that is used to transform XML documents into other types of documents, such as HTML or XML. As used herein, XSLT is a style-sheet language that uses the XML syntax to specify transformation information for XML files. An XSLT style-sheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary. An XML file 410 can be used to determine the structure and content of a blank electronic form created with the form template. The XML file 404 (with an .xsf extension), which corresponds to the solution definition 306 seen in Fig. 3, defines much of the structured editing functionality behind the form template.

[0057] Optional or auxiliary files for business logic 412 (e.g., Jscript or VBscript, graphics, XSLTs) define merge functionality, and/or other resources needed by the form template). An XML data file 402 corresponds to a Universal Resource Locator (URL) or a Universal Resource Name (URN) for the purposes of reference with respect to the electronic form template 420. A collection of default data in XML file 410 is included in the electronic form template 420 and includes data for creating a new (e.g., blank) electronic form. The one or more files of business logic 412, corresponding to business logic 310 seen in Fig. 3, can also be included in the electronic form template 420. The business logic 412 can be written in one or more languages including Java Script (JS) or

languages used for components of a data link library (DLL). As seen in Fig. 4, each file in the form template 420 is referenced from and to the form definition 404.

[0058] As with HTML files, opening an .xml file in the Microsoft® Windows® operating system will automatically launch the application that created the file. If the Microsoft® Windows® operating system cannot detect the application that created the file, it will launch the file in the default application registered for the XML file extension. When an XML file is created or edited using the electronic forms application 122, the electronic forms application 122 creates an XML processing instruction (PI) at the beginning of that XML file, which indicates that the document should be edited specifically with the electronic forms application 122. Advantageously, the PI is part of the XML standard and does not interfere with the schema on which the XML file may be based. XML files generated by the electronic forms application 122 include the XML processing instruction that identifies the corresponding template using either a URL or a URN.

[0059] Fig. 5 shows an exemplary architecture 500 for the electronic forms application 122 described herein. The architecture 500 includes a solution design component 502 for building a solution corresponding to a data file for which an electronic form can be used, an XML runtime component 504 to enter and view data in the electronic form, and a several exemplary XML solutions 506. Each of the components of the architecture 500 will now be discussed.

[0060] The solution design component 502 of the architecture 500, such as is seen by XML solution 302 in Fig. 3, allows a solution to be built. The solution design component 502 provides a user interface (UI) to handle all the design requirements for common XML solutions. The result of the solution design component 502 is the set of files that

represent the XML solution 302. The structure of the XML solution 302 in the electronic forms application 122 declaratively defines the output of the solution design component 502. Included in the solution design component 502 are an XSL editor and solution builder 510. Any script editor can be used to edit business logic script used in the electronic form. The supporting files 512 communicate with one or more application files 508 that are useful in building a solution for a data file.

[0061] In one implementation, the solution design component 502 provides a WYSIWYG forms designer and editor based on XML standards that can be used for generic XML schemas. As such, XSL editor and solution builder 510 need not be characterized as including an XML editor. Moreover, notepad 514 and support files 512 need not be present.

[0062] The runtime component 504 includes an editor frame 520 that includes XML editing 522. The XML editing 522 can function similarly to the electronic forms application 122. The editor frame 520 bidirectionally communicates with a solution infrastructure 524, such as XML solution 302 seen in Fig. 3. Each of the solution infrastructure 524 and the XML store 516 bidirectionally communicates with one of more XML documents 530. Additionally, the solution infrastructure 524 communicates with the one or more application files 508. As seen in Fig. 3, the XML document 304 points to the solution definition 306 that should process the XML document 304 on the computer 112. When the user uses the computer 112 to navigate to the XML document 304, the solution infrastructure 524 loads the required the solution definition 306. If needed, the solution definition 306 handles any contextual user interfaces (UI), runs business logic associated with the XML document 304 (e.g., business logic 310, 412), and enforces security for all computer 112 operations.

[0063] It is an option in some implementations that the XML solution infrastructure 524 works with the local XML store 526. The optional local XML store 526 can provide electronic mail (e-mail) capabilities, such as an "inbox" and a "sent items folder" for XML payloads, and to enable the ordering, filtering and aggregation of XML data that is shredded or parsed in the local XML store 526. Absent the foregoing option, XML store 526 can be omitted from XML runtime component 504.

[0064] The XML solution infrastructure 524 allows a user of computer 112 to access various XML data sources on computer 112, in an intranet, as well as on an extranet or the World Wide Web. Given the foregoing, XML Documents 530 can be displayed and edited using the XML Editing 522 of the editor frame 520.

[0065] The solutions 506 can be provided to a user of computer 112 as part of the architecture 500, where the user would like to see sample or exemplary solutions from which the user can learn about the electronic forms application 122. Solutions 506 can provide the user with a guide for customizing electronic forms and for building new solutions based on the exemplary solutions.

[0066] Figs. 6a and 6b provide respective processes 600A, 600B by which a user of workstation 602 can be provided with a solution having a corresponding electronic form that can be used by a user via an electronic forms application 618. The electronic forms application 618 and the workstation 602 seen in Figs. 6a-6b can be similar to the electronic forms application 122 and the computer 112, respectively, as seen in Fig. 1. For instance, one of the form templates 420 seen in Fig. 4 can be deployed to workstation 602 so that the user of workstation 602 can fill out the electronic form that corresponds to the form template 420. As discussed with respect to Fig. 4, the form template 420 includes the XML schema that is to be used, the formatting or presentation of the

electronic form, and any logic that the electronic form uses. The deployment of the form template 420 is available by process 600A and process 600B.

[0067] In process 600A, seen in Fig. 6a, the form template 420 is deployed to a HTTP server 610 (e.g., a Web Server). This deployment of the form template 420 enables a transparent web deployment and maintenance model. Specifically, at block 602 of Fig. 6A, a user opens a form of a certain type for the first time via an open request 604 for an XML document 606 using a URL 608 for the corresponding solution. The previously stored corresponding form template 420 is deployed from HTTP server 610 at process flow 614. The deployed form template 420 is automatically downloaded on a network and stored as an "*.XSN" file on the workstation 602 being used by the user. The downloaded form template 420 allows the user to use the form template 420 even when the workstation 602 is not connected to the network. Assuming the user has network connectivity, whenever the user opens a form, the electronic forms application 618 can be configured to check to see if a newer version of the corresponding form template 420 is available at process flow 614. If so, the newer version can be automatically downloaded to and stored on the users' workstation 602 at process flow 614.

[0068] Process 600B, seen in Fig. 6b, is an alternative to process 600A in that the form template 420 can be deployed by process flow 624 directly to workstation 602 from an information technology administrator 626 in such a way that the form template 420 will have access to local system resources and/or applications. In this case, the deployed form template 420 can be packaged for execution via process flow 624 (e.g., a "*.exe" or "*.msi" file). As seen in Fig. 6b, the workstation 602 navigates to an XML file 606 and issues an open file request at process flow 604. A URN is returned to workstation 602 at process flow 620. Here, for instance, the form template 420 can access a directory

service to obtain a users' role in an organization, where users are required to be at a certain management level to approve an electronic form, such as a travel itinerary, a purchase order or other document used in the ordinary course of business. Once obtained, the electronic forms application 618 can use this information to execute the appropriate business logic for the purchase order. The form template 420 may be deployed, for instance, along with other client code as part of a larger client deployment scenario.

[0069] Techniques for Silent Discovery and Deployment of A Solution For A Data File

Overview

Fig. 7 shows a process 700 for silently discovering and deploying a data file's solution. The process 700 is illustrated as a series of blocks representing individual operations or acts performed by the architecture 100. The process 700 may be implemented in any suitable hardware, software, firmware, or combination thereof. In the case of software and firmware, the process 700 represents a set of operations implemented as computer-executable instructions stored in memory and executable by one or more processors.

[0070] Silent Discovery and Deployment

At block 702, the system 102 receives input from a user to open the data file 126. The user may simply click on an icon representing the data file 126 or otherwise select the data file 126 after which the system 102 opens the data file 126.

[0071] At block 704, the system 102 discovers a solution identifier in the selected data file 126. This assumes that the data file 126 is one in which the electronic forms application 122 is capable of reading. The electronic forms application 122 can read data files created at some previous time by the user's or another's electronic forms application 122. In one implementation, the electronic forms application 122 can also read the data

file 126 if it is created by another application that builds a solution identifier into the data file 126. This solution identifier can give the system 102 an original source for the solution 124. The solution identifier is typically a URL (Uniform Resource Locator) or URN (Uniform Resource Name), but can include other types of names and/or locators. URLs give locations and URNs give names of resources, such as the solution 124, which are typically accessible through the communications network 104. With the solution identifier, the system 102 can determine the original source for the solution 124 (where it first came from) and whether or not the system 102 has seen the solution 124 before.

[0072] In one implementation, the solution identifier is part of a processing instruction (PI) included within the data file 126. This PI is often part of data files and can include various instructions to host applications, such as the electronic forms application 122. PIs, while not strictly data, do not rise to the level of an applet or application typically included in a solution 124 for a data file 126. For data files written in XML, for instance, the PIs are usually not written in XML, but rather are just a piece of information commonly included. A PI in an XML data file can look like

"<? eForm solutionVersion="99.9" PIVersion="99.1" href="http://insidewebsite/Forms/template.xsn" ?>".

This PI gives the electronic forms application 122 a solution identifier, which here gives the original source for the solution 124 for the data file 126. This solution identifier includes a URL indicating that the original location for the solution 124 is at a remote server accessible by accessing the communications network 104 through the network interface 118.

[0073] In another implementation, the electronic forms application 122 determines a solution that is to be used with XML data, such as the data file 126. The solution is

determined by locating a PI within the XML data. When the PI is located in the XML data, the payload of the PI is examined to determine the solution that is to be used with the XML data. A Universal Resource Locator (URL) in the PI can be used to identify a location of the solution. The solution can then be retrieved from its location. Once retrieved, the solution can be used to present a visual appearance of the XML data, to receive interactive edits to the XML data from an editing user, and then to update both the visual appearance and the XML data using the edits to the XML data from the editing user. An example of a PI in the XML data that uses a URL for discovery of its corresponding solution is:

> "<?mso-InfoPathSolution href="URL to solution"
> solutionVersion="version number"?>"
>
> where:
>
> href is a URL to the solution;
> solutionVersion is the version number of the solution used to generate the XML data (e.g., XML document).

[0074] Another example of a PI that identifies its solution by location is:

> "<?mso-infoPathSolution
> href="file:///\\ndpublic\public\user-alias\Solution\manifest.xsf"
> PIVersion="1.0.0.0"
> solutionVersion="1.0.0.90"
> productVersion="11.0.4920" ?>"

In this example, the PI has five (5) parts. The first part is the name of the PI which is 'mso-infoPathSolution'. The second part is the 'href' that is a URL that specifies the location of the solution. Stated otherwise, the 'href' is a pseudo-attribute of the PI that contains a URL that specifies the location of the solution. The URL, in different alternatives, could refer to an 'http', to a file, or to an 'ftp', in accordance with known file path conventions. Additionally, relative URLs may also be used. The third part is the version of the PI ('PIVersion'). The PIVersion pseudo-attribute is used to identify the

version of the PI itself. The fourth part is the 'solutionVersion' which is a pseudo-attribute used to identify the version number of the solution used to generate the corresponding XML document. The fifth part is the productVersion pseudo-attribute that is used to identify the version of the electronic forms application 122 that created the XML document for which the solution is to be used. In this example, the electronic forms application 122 is the InfoPath™ software application provided by Microsoft Corporation of Redmond, WA, USA. Also in this example, the third through fifth parts have the format of 'x.x.x.x'.

[0075] The foregoing implementations provide two examples in which a URL in the PI could be used to identify a location of the solution to be used with an XML document. In yet another implementation, a PI in XML data contains a name from which the solution for the XML data can be discovered. Stated otherwise, the PI in the XML data identifies its solution by name. In this case, the electronic forms application 122 can determine the solution to use with XML data (e.g., the payload) by the schema of the top level node of the structured nodes in the corresponding XML data. As in prior implementations, the electronic forms application 122 performs a process in which the PI is located and examined. An example a PI that identifies a solution by name is:

```
"<?mso-infoPathSolution
name="urn:schemas-microsoft-com:office:infopath:oob:AbsenceRequest:1033"
PIVersion="1.0.0.0"
solutionVersion="1.0.0.0"
productVersion="11.0.4920" ?>"
```

In this example, the PI has five (5) parts. The first part is the name of the PI which is 'mso-infoPathSolution'. The second part is a name pseudo-attribute that refers to a URN which is the name of the solution. Such a solution is typically registered with the electronic forms application 122 before the document is opened, and the electronic forms

application 122 looks it up by name in its list of available solutions. The third through fifth parts are identical to the immediately preceding example. As in the immediately preceding example, the present example also represents that the electronic forms application 122 is the InfoPath[tm] software application provided by Microsoft Corporation of Redmond, WA, USA.

[0076] In a still further implementation, the electronic forms application 122 can determine an online sandboxed solution that is to be used with an XML document. To do so, a PI in the XML document is located. An example of a PI for an online sandboxed solution is:

"<?mso-xDocsSolution
version="1.0.0.20"
href="http://po.car-company.com/po/po.xsf"?>"

In this example, the href contains a solution identifier for which the solution has a path with the suffix '.xsf".

[0077] In yet another implementation, the electronic forms application 122 can determine a single file online sandboxed solution that is to be used with an XML document. To do so, a PI in the XML document is located. An example of a PI for a single file online sandboxed solution is:

"<?mso-xDocsSolution version="1.0.0.20"
href="http://po.car-company.com/po/po.xsn"?>"

In this example, the href contains a solution identifier for which the solution has a path with the suffix '.xsn".

[0078] In another implementation, the electronic forms application 122 can determine the solution that is to be used with a preexisting electronic form template. To do so, a PI in the XML document is located. An example of a PI for such a case is:

"<?mso-xDocsSolution
name="urn:schemas-microsoft-
com:office:InfoPath:oob:C9956DCEE66F8924AA31D467D4F366DC"
version="1.0.0.20"?>"

In this example, the URN in the 'name' attribute is preceded by "InfoPath:oob" to designate that the template is provided with the InfoPath[tm] software application provided by Microsoft Corporation of Redmond, WA, USA.

[0079] In a still further implementation, the electronic forms application 122 can determine a trusted solution that is to be used with XML data. To do so, a PI in the XML data is located. An example of a PI for a trusted solution is:

"<?mso-xDocsSolution
name="urn:schemas-microsoft-com:office: po. car-company.com"
version="1.0.0.20"?>"

In this case, the electronic forms application 122 can determine the solution to use with XML data (e.g., the payload) by the schema of the top level node of the structured nodes in the corresponding XML data. The solution is 'trusted' in the sense that a logical relationship is established between domains to allow pass-through authentication, in which a trusting domain honors the logon authentications of a trusted domain.

[0080] One of the advantages of the electronic forms application 122 is that it enables a user to open the data file 126 without the user needing to discover the data file's solution 124, install the solution 124, or even know that the solution 124 exists. This enables users to open data files simply and easily, and in many cases enables them to edit a data file offline that they would otherwise not have been able to edit.

[0081] Continuing now with the description of process 700 in Figure 7, at block 706, the system 102 computes a special name for the solution 124 with the solution identifier. This special name is designed to be a name easily found only by the electronic forms

application 122. The special name, because it is computed and findable by the electronic forms application 122 but is not intended to be discoverable by other applications, allows for greater security in downloading possibly hostile solutions from the communications network 104.

[0082] In one implementation, the electronic forms application 122 takes the solution identifier and computes a unique special name for the solution identifier. This unique special name is repeatable. The next time, the electronic forms application 122 computes a unique special name for the same solution identifier, the same unique special name will be created. By so doing, the electronic forms application 122 can find a previously downloaded solution by computing the unique, special name and then searching for the unique, special name to determine if the solution is available locally for offline use (such as by having the solution stored in the memory 116).

[0083] In another implementation, the electronic forms application 122 computes a unique special name by computing a hash, such as a Message Digest 5 hash (MD5 hash), of the solution identifier. By computing a one-way hash of the solution identifier, the electronic forms application 122 creates a unique, special name that is a file of 128 bits from the digits of the solution identifier. The MD5 hash can be computed by knowing the URL. In one implementation, a cache for a solution can be structured as:

> %AppData%\InfoPath\Solution Cache\[16 character random GUID]\[MD5 hash of URL or URN

, where the random GUID protects the cache from other applications.

[0084] The system 102 uses the special name, which corresponds to a solution identifier and thus the data file 126's solution 124, to search through locally accessible sources for the solution 124 (block 708). The system 102 may, for instance, search in the memory

116 of Fig. 1 for files and/or folders with the same name as the special name computed in the block 706.

[0085] When the Special Name is Found

If the system 102 finds the special name (i.e., the "Yes" branch from block 710), then the solution 124 was saved earlier in the system 102 that was searched locally in the block 708. Thus, when the special name is found, the system 102 knows that the solution 124 referred to in the data file 126 (which the user is attempting to open) is accessible offline by the system 102. The solution 124 is usually stored in the memory 116 but can be stored in other searchable, local sources that the system 102 does not have to go online to find.

[0086] The solution 124, stored at the source and found using the special name, may not be current, however. Because of this, the system 102 determines whether or not the system 102 is online or offline (block 712). If online (i.e., the "Yes" branch from block 712), the system 102 will attempt to determine whether or not a more up-to-date solution should be installed (discussed below). If offline and if the locally stored solution 124 is new, then the system 102 will proceed to install the locally stored solution 124 (block 714).

[0087] If the Solution is Found and the System is Offline

If the solution 124 is found and the system 102 is offline, the system 102 proceeds to install the solution 124 from the memory 116 or another locally accessible source (block 714).

[0088] The system 102 installs the solution 124 silently in that the user does not need to know that the solution 124 was discovered, found, or was being installed. Thus, the

system 102 enables a user to edit the data file 126 when offline by silently discovering and deploying the data file's solution 124.

[0089] In one implementation, the system 102 installs the solution 124 and then opens the data file 126 in such a manner as to mimic how the data file 126 would be opened had the user opened the data file 126 with the solution accessible online, such as through opening the data file 126 with Microsoft® Internet Explorer®. The system 102 does so to make opening and editing the data file 126 as comfortable for the user as possible, because many users are familiar with opening data files online. One possible difference, however, is that if the system 102 has a slow connection to the communications network 104, the electronic forms application 122, by installing the solution 124 from a local source like the memory 116, may more quickly open the data file 126 than if the user were online.

[0090] In block 716, the system 102 opens the data file 126 to enable the user to edit the data file 126. One example of an opened data file (and solution) enabling edits is the travel itinerary form 200 of Fig. 2. In this example, the user is able to edit the data file 126 by adding, deleting, or changing data in data entry fields (like the data-entry field 206 even though offline.

[0091] Following the previous blocks, a user can easily open a data file offline without having to discover or deploy the data file's solution. This enables users, for example, after first opening a solution online, to open a data file offline. A user can open a data file online and edit it by adding a date through the data-entry field 202 of the travel itinerary form 200 and then stop editing the data file (the data file would contain the added date by the system 102 adding the date to the data file). The user could then go offline, such as by taking his or her laptop on a business trip, and complete filling out the

electronic form. Or the user could send the partially filled-out data file to another user to fill out the rest of the electronic form, which the other user could do so long as the other user's system contains a stored solution. This flexibility allows users and businesses a greater ability to use information by keeping data and solutions separate and by allowing offline use of data files.

[0092] If the Solution is Found and the System is Online

Assuming the system 102 finds the special name and the system is online, the system 102 will attempt to determine whether the current solution is the most recent version or a more up-to-date solution is available. In block 718, the system 102 compares the time stamp of the stored solution 124 and the online solution. Since the system 102 is online, it can access the solution (here we assume that the origin of the solution 124 is from an online source). If the solution identifier from the data file 126 selected by the user contains a reference to the solution 124 being accessible online, the system 102 goes online to check whether or not the online solution is newer than the stored solution 124 (block 720). In one implementation, the system 102 compares a time stamp of the online solution with a time stamp on the stored solution 124.

[0093] If the online solution is not newer (i.e., the "No" branch from block 720), the system 102 proceeds to the block 714, installing the stored solution 124. If the online solution is newer than the stored solution 124 (i.e., the "Yes" branch from block 720), the system 102 either replaces the stored solution 124 with the online solution or otherwise updates the older, stored solution 124.

[0094] Downloading the Solution for Later Use

In block 722, the architecture 100 (or the system 102 by accessing the communications network 104) downloads a solution into a locally accessible source such

as the memory 116. The system 102 downloads this solution when the data file 126 selected by a user contains a solution identifier for a solution for which the system 102 does not have local access (such as it not being cached) or for which the system 102 has local access but the cached or stored version of the solution (the solution 124) is older than the online version.

[0095] In either case, the system 102 has already discovered the solution identifier for the solution and computed a special name for the solution. The system 102 then downloads the solution from the online source. Note, however, that if system 102 is offline, then process 700 will terminate in a failure mode. If system 102 is online, then process 700 proceeds from block 722 to block 724 and saves the solution into a folder named with the special name (block 724). If a solution already exists in that folder, the system 102 replaces it with the newer version or otherwise updates the currently cached solution. The resulting new or updated version will then be in the solution 124.

[0096] In one implementation, the system 102 saves the solution to a unique location within the system 102's accessible memory. The system 102 does so in cases where the system 102 is used by multiple users. By so doing, the system 102 is able to determine which of the users that use the system 102 or load files into memory locally accessible by the system 102 saved the particular solution. Also by so doing, the system 102 may provide greater security for the computer 112 and its users.

[0097] Process 700 in Fig. 7 allowed for editing of data files while online or offline. In each of the various implementations discussed above in the context of Fig. 7, the electronic forms application 122 performed a process in which a PI is located and examined. Stated otherwise, the electronic forms application 122 performs a process that searches XML data to find a solution of the XML data. In other implementations, the

search by electronic forms application 122 may examine various storage locations for the solution, such as file storage on a network resource that is in communication with a computing device that is executing the electronic forms application 122. Alternatively, the search may examine local cache in a computing device that is executing the electronic forms application 122. If the search by the electronic forms application 122 in the XML data does not locate a PI that specifically identifies the electronic forms application 122, then a diagnostic can be output, such as by way of a visibly displayed descriptive error. If a PI is found in the search that specifically identifies the electronic forms application 122, then it is determined if the PI contains a 'href' attribute. If the 'href' attribute is in the PI, then a computation is made of the name of the folder in the solution cache using the URL associated with the 'href' attribute. A look up is first performed on the list of locally cached solutions using the URL in the 'href' attribute (e.g., where the solution is 'online sandboxed'). If, however, the solution is not located in the list of locally cached solutions, then the solution must be retrieved from another resource. In this case, the solution can be downloaded from the URL associated with the 'href' attribute in PI.

[0098] In another implementation, where the solution is located in the list of locally cached solutions, it can be ascertained whether the most recent version of the solution has been obtained. To do, a version, electronic tag, or time stamp in the PI can be examined, such as by an auto-update process. If the examination indicates a change in the original solution (e.g., a server copy of the solution is different), then the latest solution can be downloaded. For instance, the electronic forms application 122 can be configured to explode the solution to a temporary location within the locally cached solutions and then can compare the version of the downloaded solution to the version of the current cached copy of the solution.

[0099] If the search by the electronic forms application 122 finds a PI but fails to locate the solution in the list of locally cached solutions or by download, then a diagnostic can be output, such as by way of a visibly displayed descriptive error. If the search by the electronic forms application 122 finds a PI that contains a name attribute associated with a URN, a computation can be performed of the name of the folder in the solution using the URN. A lookup can then be performed on a locally installed solution using the name attribute in a catalog of solutions. If the solution is thereby found, a comparison can be made between the cached version of the solution and the original version of the solution. This comparison is performed to make sure that the cached version of the solution is up-to-date. In the case of the InfoPath™ software application provided by Microsoft Corporation of Redmond, WA, USA, through the Microsoft Office™ software application, if the name is prefixed by a character string that includes "InfoPath" and "oob", then the XML data located by the application is referring to an InfoPath template or out-of-the-box (OOB) solution. In such a case, an assumption might be made that solutions for the InfoPath templates are installed at a location shared by all users of a specific computing device (e.g., C:\Program Files\Microsoft Office\Templates).

[0100] The steps set forth in process 700 in Fig. 7 for discovering and using a solution are described above for various implementations. Still further implementations can be used by the electronic forms application 122 to discover and use a solution. In these further implementations, the electronic forms application 122 attempts to associate an XML document with a solution by using a href attribute in a PI in the XML document that points to a URL. In yet other implementations, the electronic forms application 122 initiates a search in an XML document for a PI having a href attribute, a name, and a

version. When the electronic forms application 122 finds such a PI, the XML document can be associated with a solution through the use of the href attribute (e.g., a URL).

[0101] When the InfoPath™ software application provided by Microsoft Corporation of Redmond, WA, USA, through the Microsoft Office™ software application is used to design an electronic form, a PI in an XML document corresponding to the electronic form will have various characteristics. As such, any electronic forms application 122 that is attempting to discover a solution for an electronic form that was created by the InfoPath™ software application should be advantageously configured to locate the solution by searching for a PI in the corresponding XML document that has one or more of these various characteristics. When so discovered, the electronic forms application 122, which may be different than the InfoPath™ software application, can then be used to view and/or interactively enter/change XML data through the electronic form that was created by the InfoPath™ software application.

[0102] Once such characteristic of an XML document corresponding to an electronic form that was designed, created, or changed by the InfoPath™ software application is that it has a PI containing the solution identifier that will most likely be the first PI in the XML document corresponding to the electronic form. Another such characteristic is that the PI containing the solution identifier contains both a href attribute and a character string that identifying the PI version and/or the product version of the InfoPath™ software application. The href attribute need not be the first pseudo-attribute in the PI, but could be anywhere in the PI. As such, the PI version could be first in the PI followed by the href attribute. Yet another characteristic is that the target of the PI containing the solution identifier also contains the character string "mso-InfoPathSolution" solution, where that character string is the first character string in the PI and could be followed by a href

attribute. Yet another characteristic is that the PI containing the solution identifier contains a URL or URN that will most likely point to a path having a suffix that is either '*.xsf' (e.g., a manifest or listing other files) or '*.xsn' (a file that contains multiple files compressed into one file and that are extractable with the extract '*.exe' utility).

[0103] Given the foregoing characteristics of electronic forms designed, created, or changed using the InfoPath™ software application, the electronic forms application 122 can be configured to discover the solution of an electronic form by simply finding the first PI in the XML document. Alternatively, the electronic forms application 122 can be configured to discover the solution by examining a PI in the XML document for the electronic form to see if it includes a character string that is likely to represent a solution corresponding to the XML document. When the likelihood is high, the name can be used to discover the solution. As a further alternative, the electronic forms application 122 can be configured to discover the solution by examining the target in a PI in the corresponding XML document for the character string "mso-InfoPathSolution", and/or by looking for a URL in the PI having a suffix of *.xsf or *.xsn. As yet another alternative, the electronic forms application 122 can be configured to discover the solution by trying, as a potential solution, each name that is in quotation marks in the PI and is associated with a href attribute in a PI in the XML document, and then using the solution that is returned after a success following one or more of such attempts. As mentioned above, the href attribute need not be the first pseudo-attribute in the PI, and the electronic forms application 122 may take this into account when assessing the PI for its likelihood of containing the correct solution identifier. In a still further another alternative, the electronic forms application 122 can be configured to discover the solution by trying as a potential solution each URL in each PI of an XML document, and then using the solution

that is returned after a success following one or more of such attempts. As noted above, the URL or URN in the successful PI will most likely point to a '*.xsf' or '*.xsn' path, and the electronic forms application 122 may take this into account when assessing the PI for its likelihood of containing the correct solution identifier.

[0104] Data Files, Transformation Files, Rendering Files, and Rendered Forms

As discussed above, solution 124 contains presentation folder 128 that includes the rendering file 128a and the transformation file 128b. The data file 126, transformation file 128a, rendering file 128b, and a rendered form work together to allow a user to edit the data file 126. A user can input data into and view data in the data file 126 through the rendered form of the data file. This rendered form is the result of executing the rendering file 128a, which is created by applying the transformation file 128b on the data file 126.

[0105] Figs. 2 and 8 shows the rendered form 200 entitled "Travel Itinerary", which is generated by executing the rendering file 128a. This travel itinerary form 200 is rendered so as to contain data-entry fields in which a user can enter data. These data-entry fields map to the data file 126, so that the data entered into the form are retained in the data file 126.

[0106] Data input into a particular data-entry field of the rendered form is stored in a particular node of the data file 126. Data-entry fields of the rendered form correlate to nodes of the data file 126 in part because the rendered form is the result of the transformation file 128b being applied on the data file 126. The system 102 can use various ways to detect which data-entry fields correlate to which nodes of the data file 126, including through mapping with XML Path Language (XPath) expressions that

address parts of data file 126 (e.g., an XML document) by providing basic facilities for manipulation of strings, numbers and Booleans.

[0107] Also in Figs. 2 and 8, a graphical representation of the data file 126 is shown as a data file tree 202. The data file tree 202 shows icons representing nodes of the data file 126. Many of these nodes correlate to data-entry fields shown in the travel itinerary form 200 that is rendered. For instance, a trip start date node 204 correlates to the trip start date data-entry field 206. Thus, data entered by a user into the trip start date data-entry field 206 can be stored in the trip start date node 204 of the data file 126.

[0108] The transformation file 128b also correlates to the data file 126. Nodes of the data file 126 correlate to particular parts of the transformation file 128b, also called nodes for the purposes of this description. Thus, nodes of the transformation file 128b correlate to nodes of the data file 126. This correlation can arise from nodes of the transformation file 128b being mapped to the nodes of the data file 126, including through XPath expressions, or otherwise.

[0109] That certain nodes of the transformation file 128b correlate to certain nodes of the data file 126 is often not enough, however, for the system 102 to accurately reflect a change in a particular node of the data file 126 by simply applying only a particular node of the transformation file 128b on a particular node of the data file 126. A node of the transformation file 128b, when applied on a node of the data file 126, may affect many nodes of the data file 126. A node of the transformation file 128b could, for instance, be one that, as part of being applied on a node of the data file 126, is also applied on previously filled-in or as-yet-unfilled-in nodes of the data file 126. This concept is illustrated in Fig. 8.

[0110] Fig. 8 shows the travel itinerary form 200 that has been rendered, in this case including filled-in data-entry fields. Here the rendered form is generated after data was input by a user into the trip start date data-entry field 206, "03/13/2002". After the electronic forms application 122 produced a rendering file, the system 102 renders the rendering file. In this example, the transformation file 128b, when applied, affected other nodes of the data-entry field other than just the trip start date node 204, in this case an event start date node 802. Because the transformation file 128b (or a part thereof) affected the event start date node 802, the rendering file 128a included that change. Thus, when executed, the rendering file 128a renders an updated travel itinerary form 200, including the data shown in an event start date data-entry field 804 in Fig. 8. Here, the transformation file 128b altered the event start date node 802 to include the exact same data entered into the trip start date data-entry field 206 of Fig. 8. The transformation file 128b may perform such an action to make it easier for the user in cases where a future node/data-entry field is likely to have the same data.

[0111] Further, the node of the transformation file 128b may direct the system to perform computations or other operations using other resources, like a database. For these and other reasons, the electronic forms application 122 analyzes the results of nodes of the transformation file 128b being applied on nodes of the data file 126 or nodes of some hypothetical data file, which will be discussed in greater detail below.

[0112] In some implementations, the transformation file 128b is an XSLT (eXtensible Style-sheet Language Transformation) file, which, when applied to an XML data file 126, generates a XHTML (eXtensible Hyper-Text Machine Language) or HTML (Hyper-Text Machine Language) rendering file (such as the rendering file 128a). The transformation file 128b can also be an arbitrary XSLT file, such as a custom-made file or some other

W3C-compliant file. XHTML and HTML files can be used to show a view on the screen 110, such as the travel itinerary form 200 of Figs. 2 and 8.

[0113] Like transformation files, data files can come in various types and styles. Hierarchical data files can be written in XML or some other mark-up language, or can be written in other hierarchical languages. Hierarchical data files also are typically concise and data-centered so that the data they contain can be more easily accessed or manipulated by multiple software applications, including software not typically used in a solution, such as an application that searches for a particular type of data and compiles that data into a report. A non-typical application, for example, could be one that compiles a report of all of the travel itineraries filled out in electronic forms by a certain person by searching through and compiling the data entered in travel itinerary data files for a particular person.

[0114] Validating Data From A User In Real-Time

Overview

A system, such as the system 102 of Fig. 1, displays an electronic form with data-entry fields to allow a user to enter data. The user can enter data in a data-entry field and know, as he does so, whether or not the data entered is valid or invalid. By so doing, the system 102 provides an easy, intuitive, and efficient way for a user to enter and correct data intended for a structured data file.

[0115] Fig. 9 shows a process 900 for validating data entered into an electronic form in real-time. The process 900 is illustrated as a series of blocks representing individual operations or acts performed by the system 102. The process 900 may be implemented in any suitable hardware, software, firmware, or combination thereof. In the case of software and firmware, the process 900 represents a set of operations implemented as

computer-executable instructions stored in memory and executable by one or more processors.

[0116] Notifying A User of Errors in Real-Time

At block 902, the system 102 displays an electronic form having data-entry fields. The electronic form can be blank or contain filled data-entry fields. An expense report 1010 electronic form in Fig. 10 is an example of an electronic form that contains data in data-entry fields.

[0117] The system 102 displays an electronic form in a manner aimed at making a user comfortable with editing the electronic form. It can do so by presenting the electronic form with user-friendly features like those used in popular word-processing programs, such as Microsoft® Word®. Certain features, like undoing previous entries on command, advancing from one data-entry field to another by clicking on the data-entry field or tabbing from the prior data-entry field, cut-and-paste abilities, and similar features are included to enhance a user's data-entry experience. For example, the system 102 displays an electronic form having some of these features in Fig. 10, the expense report 1010 electronic form.

[0118] At block 904, with the electronic form presented to the user, the system 102 enables the user to input data into a data-entry field. The user can type in data, cut-and-paste it from another source, and otherwise enter data into the fields. The user can use the user-input devices 106, including a keyboard 110 and other device(s) (such as a touch screen, track ball, voice-activation, and the like). In Fig. 10, for example, the user enters "1/27/2003" into the report date data-entry field 1012 of the expense report 1010.

[0119] At block 906, the system 102 receives the data entered into the data-entry field 1012 by the user. The system 102 receives the data from the user through the user-input

devices 106 and the user interface 134 (both of Fig. 1). The system 102 can receive the data character-by-character, when the data-entry field is full, or when the user attempts to continue, such as by tabbing to move to another data-entry field. In the foregoing example, the system 102 receives "1/27/2003" from the user when the user attempts to advance to the next data-entry field.

[0120] At block 908, the system 102 validates the data received into the data-entry field in the electronic form. By using validation rules stored in the logic file 130 of the solution 124 and a real-time validation tool 136 stored in memory 116, the system 102 can analyze the data to determine if it is valid. In an alternative implementation, the real-time validation tool 136 can be included as a part of the solution 124 in memory 116. The real-time validation tool 136 refers to the validation rules, if any, in the logic file 120 governing that particular data-entry field (in this example the report date data-entry field 1012). The real-time validation tool 136 validates the data entered into a data-entry field without the user having to save or submit the electronic form. It can do so by applying validation rules associated with the node of the structured data file corresponding to data-entry field into which the data was entered.

[0121] The real-time validation tool 136 can apply validation rules from many different sources. One source for validation rules is a schema governing the structured data file. Other sources of validation rules can include preset and script-based custom validation rules.

[0122] For script-based custom validation rules, the real-time validation tool 136 enables these rules to refer to multiple nodes in a structured data file, including nodes governing or governed by other nodes. Thus, the real-time validation tool 136 can validate data from a data-entry field intended for a particular node by checking validation rules

associated with that particular node. Through so doing, the real-time validation tool 136 can validate data entered into one node of a group with the validation rules governing the group of which the node is a part.

[0123] For example, if a group of nodes contains four nodes, and is associated with a script-based validation rule requiring that the total for the data in all of the four nodes not exceed 1000, the real-time validation tool 136 can validate each node against this rule. Thus, if the first node contains 100, the second 400, and the third 300, the real-time validation tool 136 will find the data intended for the fourth node invalid if it is greater than 200 (because 100+400+300+200=1000).

[0124] In some cases the real-time validation tool 136 can build validation rules from a schema containing logic that governs a structured data file. This logic sets forth the bounds of what data nodes in a structured data file can contain, or the structure the nodes should have. Data entered into a structured data file can violate this logic, making the structured data file invalid. This invalid data may cause a structural error or a data-type error in the structured data file, possibly making the structured data file useless. To combat this, the real-time validation tool 136 can build validation rules from a structured data file's schema.

[0125] Because structural errors are especially important, the real-time validation tool 136 treats these types of errors seriously. To make sure that a user treats these errors seriously, the real-time validation tool 136 builds validation rules for structural errors that stop a user from continuing to edit an electronic form if the real-time validation tool 136 detects a structural error. Validation rules that stop the user from continuing to edit the electronic form (except for fixing that invalid data) are called modal validation rules, and errors that violate them, modal errors.

[0126] For less serious errors, such as data-type errors, the real-time validation tool 136 builds validation rules that do not stop the user from continuing. These are called modeless validation rules, and errors that violate them, modeless errors.

[0127] To aid the real-time validation tool 136 in validating data in real-time, validation rules are associated with particular nodes. By so doing, with each new piece of data received, the real-time validation tool 136 is capable of comparing the data received against an appropriate list of validation rules associated with the node for which the data received is intended. Because this list of validation rules can be very short for each particular node, the real-time validation tool 136 has fewer validation rules to check for each piece of data entered than if it had to check all the validation rules for the node's structured data file. This speeds up the process of validation.

[0128] Continuing the previous example, at the block 908 the system 102 validates the data entered, "1/27/2003", against validation rules associated with the report date data-entry field 1012, thereby determining if the data entered is valid.

[0129] In block 910 the system 102 determines whether to proceed to block 914 or 912 depending on whether the data is valid. If the real-time validation tool 136 determines that the data entered is not valid, it proceeds to the block 914, discussed below. If, on the other hand, the real-time validation tool 136 determines it to be valid, the system 102 continues to block 912, allowing the user to continue editing the electronic form. Continuing the ongoing example, if the real-time validation tool 136 determines that the data "1/27/2003" is valid, the system 102 continues on to the block 912. If not, it proceeds to block 914.

[0130] At the block 912, the system 102 enables the user to input data into another data-entry field. In Fig. 10, for example, it would allow the user to proceed to enter data into

the expense period data-entry field 1014 after the data entered into the report date data-entry field 1012 was determined to be valid. The system 102 can allow the user to proceed to another data-entry field as well, depending on the user's preference.

[0131] If the data is invalid, the system 102 proceeds to the block 914. At the block 914 the system 102, through the real-time validation tool 136, determines whether to proceed to block 916 if the error is not modal and 918 if it is.

[0132] Continuing the previous example, assume that the data entered into the report date data-entry field 1012 is invalid. Assume also that "1/27/2003" is not defined to be a modal error. (Modal errors are those for which the real-time validation tool 136 rolls back the invalid entry requiring the user to re-enter another entry before continuing on to edit another data-entry field or requires the user to correct.) Thus, in this example, "1/27/2003", is invalid, but is a modeless error.

[0133] In the block 916, the real-time validation tool 136 alerts the user of a modeless error by marking the data-entry field as containing an error, but allows the user to continue editing the electronic form. To make the editing process as easy, intuitive, and efficient as possible, the real-time validation tool 136 can mark the data-entry field from which the invalid error was entered in many helpful ways. Optionally, or in combination with the foregoing, a prompt or diagnostic can be displayed. The real-time validation tool 136 can highlight the error in the data-entry field, including with a red box, a dashed red box, a colored underline, a squiggly underline, shading, and the like. The real-time validation tool 136 can also alert the user with a dialog box in a pop-up window, either automatically or only if the user asks for information about the error.

[0134] The real-time validation tool 136, for example, can present a dialog box or other presentation manner explaining the error or what type of data is required by the data-

entry field. The real-time validation tool 136 can present a short comment that disappears quickly or is only shown if the user moves his cursor or mouse pointer over the data-entry field. The real-time validation tool 136 can also provide additional information on request. Many manners of showing the user that the data is invalid as well as showing information about the error can be used. These ways of notifying the user can be chosen by a developer when creating a custom validation rule. For modeless errors, the real-time validation tool 136 permits the user to proceed, according to the block 912, discussed above. For modal errors, however, the real-time validation tool 136 can present a dialog (block 918). The user then can dismiss the dialog. Once the dialog is dismissed, the real-time validation tool 136 rolls back the invalid entry and enables the user to continue editing the electronic form. This editing can include re-inputting data into the data-entry field (block 920), or editing another data-entry field. Alternatively, the real-time validation tool 136 leaves the error in the document, but will not allow the user to continue editing the document without first correcting the error.

[0135] In the block 918, the real-time validation tool 136 presents an alert to notify the user of the invalid entry. This alert is intended to inform the user that the error is important and must be fixed. The alert does not have to be a pop-up window, but should be obvious enough to provide the user with an easy-to-notice notification that the user has entered data causing an error. The alert, in one implementation, is a pop-up window that requires the user to pause in editing the electronic form by making the user click on an "OK" button in the alert. This stops the user mentally, helping the user to notice that he must fix the data-entry field having the error before proceeding. The alert can contain no, little, or extensive information about the error. The information can be presented automatically or after the system 102 receives a request for the information.

[0136] Fig. 10 shows the partially filled-in expense report 1010 electronic form with a date dialog box 1002 in an alert area display and arising from invalid data causing a modal error. The dialog box contains a button marked "OK" that the user must select (a date dialog button 1004). The date dialog box 1012 also contains a date information line 1006 informing the user about the error, "The Report Date Must Be Later Than the Expense Period." This information is intended to aid the user's attempt to correct the invalid data.

[0137] After presenting the user with some sort of alert in block 918 (Fig. 9), the real-time validation tool 136 enables the user to re-input data into the data-entry field containing the modal error (block 920). Here the user must change the data within the data-entry field to a valid or modeless error before continuing to edit new data-entry fields in the electronic form. Once the user inputs new (or the same) data into the data-entry field, the system 102 receives the data at the block 906 and so forth. To proceed, the user must enter data that is not a modal error; if the user does not, the system 102 will follow the process 900, continuing to find the data modally invalid and not permit the user to continue.

[0138] Through this process 900 of Fig. 9, the system 102 can receive and validate data in real-time. By so doing, a user can easily, accurately, and efficiently edit a structured data file through entry of data into data-entry fields in an electronic form.

[0139] The example set forth in Fig. 9 is not intended to be limiting on the abilities of the system 102 or the real-time validation tool 136. Other types of forms, data-entry fields, and alerts can be used.

[0140] The above devices and applications are merely representative, and other known devices and applications may be substituted for or added to those shown in Fig. 1. One

example of another known device that can be substituted for those shown in Fig. 1 is the device shown in Fig. 11.

[0141] Exemplary Computing System and Environment.

Fig. 11 shows an exemplary computer system and environment that can be used to implement the processes described herein. A computer 1142, which can be similar to computer 112 in Fig. 1, includes one or more processors or processing units 1144, a system memory 1146, and a bus 1148 that couples various system components including the system memory 1146 to processors 1144. The bus 1148 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. The system memory 1146 includes read only memory (ROM) 1150 and random access memory (RAM) 1152. A basic input/output system (BIOS) 1154, containing the basic routines that help to transfer information between elements within computer 1142, such as during start-up, is stored in ROM 1150.

[0142] Computer 1142 further includes a hard disk drive 1156 for reading from and writing to a hard disk (not shown), a magnetic disk drive 1158 for reading from and writing to a removable magnetic disk 1160, and an optical disk drive 1162 for reading from or writing to a removable optical disk 1164 such as a CD ROM or other optical media. The hard disk drive 1156, magnetic disk drive 1158, and optical disk drive 1162 are connected to the bus 1148 by an SCSI interface 1166 or some other appropriate interface. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for computer 1142. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 1160 and a removable optical disk 1164, it should

be appreciated by those skilled in the art that other types of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, random access memories (RAMs), read only memories (ROMs), and the like, may also be used in the exemplary operating environment.

[0143] A number of program modules may be stored on the hard disk 1156, magnetic disk 1160, optical disk 1164, ROM 1150, or RAM 1152, including an operating system 1170, one or more application programs 1172 (such as the electronic forms application 112), other program modules 1174, and program data 1176. A user may enter commands and information into computer 1142 through input devices such as a keyboard 1178 and a pointing device 1180. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are connected to the processing unit 1144 through an interface 1182 that is coupled to the bus 1148. A monitor 1184 or other type of display device is also connected to the bus 1148 via an interface, such as a video adapter 1186. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

[0144] Computer 1142 commonly operates in a networked environment using logical connections to one or more remote computers, such as a remote computer 1188. The remote computer 1188 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 1142. The logical connections depicted in Fig. 5 include a local area network (LAN) 1190 and a wide area network

(WAN) 1192. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

[0145] When used in a LAN networking environment, computer 1142 is connected to the local network through a network interface or adapter 1194. When used in a WAN networking environment, computer 1142 typically includes a modem 1196 or other means for establishing communications over the wide area network 1192, such as the Internet. The modem 1196, which may be internal or external, is connected to the bus 1148 via a serial port interface 1168. In a networked environment, program modules depicted relative to the personal computer 1142, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

[0146] Generally, the data processors of computer 1142 are programmed by means of instructions stored at different times in the various computer-readable storage media of the computer. Programs and operating systems are typically distributed, for example, on floppy disks or CD-ROMs. From there, they are installed or loaded into the secondary memory of a computer. At execution, they are loaded at least partially into the computer's primary electronic memory. The invention described herein includes these and other various types of computer-readable storage media when such media contain instructions or programs for implementing the blocks described below in conjunction with a microprocessor or other data processor. The invention also includes the computer itself when programmed according to the methods and techniques described herein.

[0147] For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is

recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the data processor(s) of the computer.

[0148] Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.